springboot源码分析外部tomcat启动war包的原理

前情提示:以前我们说过，springboot程序是jar的方式,是通过IOC容器启动 带动了tomcat的启动

那么，我们把springboot的程序打成war的时候，是怎么样的原理了?(tomcat启动带动IOC容器的启动)

从疑问开始: 我们把springboot打成war的包时候，为什么要在启动类程序上实现 SpringBootServletInitializer 接口?

以及com.tuling.TulingvipSpringbootWarApplication#configure（）是在什么时候触发调用的?

```
@SpringBootApplication
public class TulingvipSpringbootWarApplication extends SpringBootServletInitializer {

        public static void main(String[] args) {
                SpringApplication.run(TulingvipSpringbootWarApplication.class, args);
        }

        @Override
        protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
                return application.sources(TulingvipSpringbootWarApplication.class);
        }

}
```

1）从servlet3.0的特性说起

8.2.4 Shared libraries / runtimes pluggability

```
The ServletContainerInitializer class is looked up via the jar services API.

For each application, an instance of the ServletContainerInitializer is created by the container at application startup time.

The framework providing an implementation of the ServletContainerInitializer MUST bundle in the META-INF/services dire

javax.servlet.ServletContainerInitializer, as per the jar services API,that points to the implementation class of the ServletC

In addition to the ServletContainerInitializer we also have an annotation -HandlesTypes.

The HandlesTypes annotation on the implementation of the ServletContainerInitializer is used to express interest in class
have annotations (type, method or field level annotations) specified in the value of
the HandlesTypes or if it extends / implements one those classes anywhere in the
class' super types.
The HandlesTypes annotation is applied irrespective of the
setting of metadata-complete.
```

1.1)web应用启动，会创建当前Web应用导入jar包中的 ServletContainerInitializer类的实例

1.2)ServletContainerInitializer 类必须放在jar包的 META-INF/services目录下,文件名称为 javax.servlet.ServletContainerInitializer

1.3)文件的内容指向ServletContainerInitializer实现类的全路径

1.4)使用@HandlesTypes 在我们应用启动的时候，加载我们感兴趣的类

```
@HandlesTypes(WebApplicationInitializer.class)
@HandlesTypes(WebApplicationInitializer.class)
public class SpringServletContainerInitializer implements ServletContainerInitializer {


	@Override
	public void onStartup(@Nullable Set<Class<?>> webAppInitializerClasses, ServletContext servletContext)
			throws ServletException {

	//创建保存感兴趣的类的集合
		List<WebApplicationInitializer> initializers = new LinkedList<>();

		if (webAppInitializerClasses != null) {
			for (Class<?> waiClass : webAppInitializerClasses) {
				// Be defensive: Some servlet containers provide us with invalid classes,
				// no matter what @HandlesTypes says...
				//判断感兴趣的类不是接口不是抽象类
				if (!waiClass.isInterface() && !Modifier.isAbstract(waiClass.getModifiers()) &&
						WebApplicationInitializer.class.isAssignableFrom(waiClass)) {
					try {
					  //通过反射创建实例并且加入到集合中
						initializers.add((WebApplicationInitializer)
								ReflectionUtils.accessibleConstructor(waiClass).newInstance());
					}
					catch (Throwable ex) {
						throw new ServletException("Failed to instantiate WebApplicationInitializer class", ex);
					}
				}
			}
		}

		if (initializers.isEmpty()) {
			servletContext.log("No Spring WebApplicationInitializer types detected on classpath");
			return;
		}

		servletContext.log(initializers.size() + " Spring WebApplicationInitializers detected on classpath");
		AnnotationAwareOrderComparator.sort(initializers);

		//循环调用集合中的感兴趣类对象的onstartup的方法
		for (WebApplicationInitializer initializer : initializers) {
			initializer.onStartup(servletContext);
		}
	}

}
```

主要流程:

 1)tomcat启动，然后去org\springframework\spring-web\5.0.10.RELEASE\spring-web-5.0.10.RELEASE.jar!\META-INF\services\javax.servlet.ServletContainerInitializer 文件中

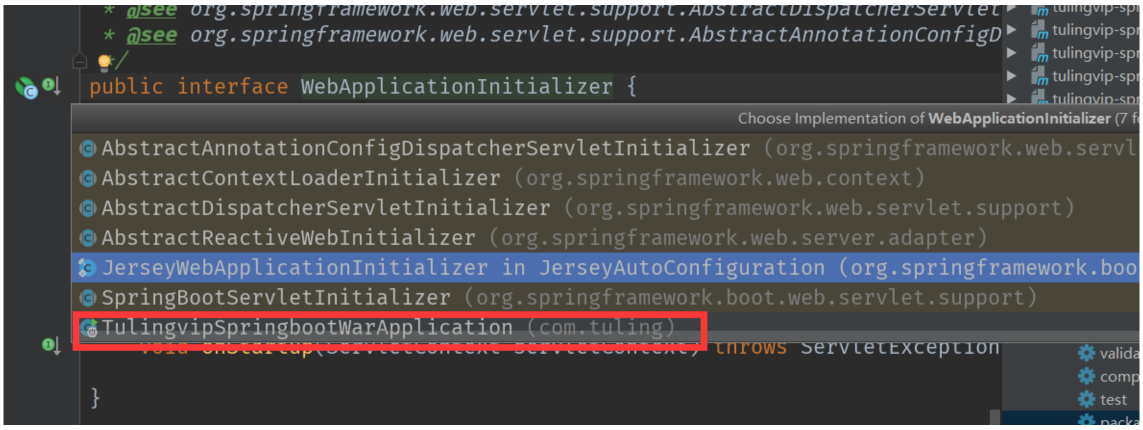
org.springframework.web.SpringServletContainerInitializer的实例

2)讲@HandlesTypes标注的感兴趣的类（WebApplicationInitializer）都传入到 osStartup()的方法中Set<Class<?>>参数中

为这些感兴趣的类创建实例　ReflectionUtils.accessibleConstructor(waiClass).newInstance();

3)调用WebApplicationInitializer的onStartup方法

3.1)我们来看下 WebApplicationInitializer的实现类就有我们的TulingVipSpringBootWarApplication类



```
 * @see org.springframework.web.servlet.support.AbstractDispatcherServlet
 * @see org.springframework.web.servlet.support.AbstractAnnotationConfigD
 */
public interface WebApplicationInitializer {
```

Choose Implementation of **WebApplicationInitializer** (7 f

```
AbstractAnnotationConfigDispatcherServletInitializer (org.springframework.web.servl
AbstractContextLoaderInitializer (org.springframework.web.context)
AbstractDispatcherServletInitializer (org.springframework.web.servlet.support)
AbstractReactiveWebInitializer (org.springframework.web.server.adapter)
JerseyWebApplicationInitializer in JerseyAutoConfiguration (org.springframework.boo
SpringBootServletInitializer (org.springframework.boot.web.servlet.support)
TulingvipSpringbootWarApplication (com.tuling)
    void onStartup(ServletContext servletContext) throws ServletException
}
```

3.2) 由于TulingVipSpringBootWarApplication这个类 没有自己的onstart方法，那么就调用父类SpringBootServletInitializer的onStartup的方法

```
    public void onStartup(ServletContext servletContext) throws ServletException {

        this.logger = LogFactory.getLog(getClass());
        //创建web应用的上下文对象
        WebApplicationContext rootAppContext = createRootApplicationContext(
                servletContext);
        if (rootAppContext != null) {
            servletContext.addListener(new ContextLoaderListener(rootAppContext) {
                @Override
                public void contextInitialized(ServletContextEvent event) {
                    // no-op because the application context is already initialized
                }
            });
        }
        else {
            this.logger.debug("No ContextLoaderListener registered, as "
                    + "createRootApplicationContext() did not "
                    + "return an application context");
        }
    }


//////      //创建web应用的上下文对象
protected WebApplicationContext createRootApplicationContext(ServletContext servletContext) {
    //创建spring应用的构建器
        SpringApplicationBuilder builder = createSpringApplicationBuilder();
        builder.main(getClass());
        //设置环境
        ApplicationContext parent = getExistingRootWebApplicationContext(servletContext);
        if (parent != null) {
            this.logger.info("Root context already created (using as parent).");
            servletContext.setAttribute(
                    WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, null);
            builder.initializers(new ParentContextApplicationContextInitializer(parent));
        }
        builder.initializers(new ServletContextApplicationContextInitializer(servletContext));
        builder.contextClass(AnnotationConfigServletWebServerApplicationContext.class);

        //调用我们自己启动类上的confiure方法 传入我们自己的主启动类
        builder = configure(builder);
        builder.listeners(new WebEnvironmentPropertySourceInitializer(servletContext));
        SpringApplication application = builder.build();
        if (application.getAllSources().isEmpty() && AnnotationUtils
                .findAnnotation(getClass(), Configuration.class) != null) {
            application.addPrimarySources(Collections.singleton(getClass()));
        }
        Assert.state(!application.getAllSources().isEmpty(),
                "No SpringApplication sources have been defined. Either override the "
                        + "configure method or add an @Configuration annotation");
        // Ensure error pages are registered
        if (this.registerErrorPageFilter) {
            application.addPrimarySources(
                    Collections.singleton(ErrorPageFilterConfiguration.class));
        }
        //调用我们类上的run方法
        return run(application);
    }


public ConfigurableApplicationContext run(String... args) {
        StopWatch stopWatch = new StopWatch();
        stopWatch.start();
        ConfigurableApplicationContext context = null;
        Collection<SpringBootExceptionReporter> exceptionReporters = new ArrayList<>();
        configureHeadlessProperty();
        SpringApplicationRunListeners listeners = getRunListeners(args);
        listeners.starting();
        try {
            ApplicationArguments applicationArguments = new DefaultApplicationArguments(
                    args);
            ConfigurableEnvironment environment = prepareEnvironment(listeners,
                    applicationArguments);
            configureIgnoreBeanInfo(environment);
            Banner printedBanner = printBanner(environment);
            context = createApplicationContext();
            exceptionReporters = getSpringFactoriesInstances(
                    SpringBootExceptionReporter.class,
                    new Class[] { ConfigurableApplicationContext.class }, context);
            prepareContext(context, environment, listeners, applicationArguments,printedBanner);

            //启动我们的IOC 容器
            refreshContext(context);

            afterRefresh(context, applicationArguments);
            stopWatch.stop();
            if (this.logStartupInfo) {
                new StartupInfoLogger(this.mainApplicationClass)
                        .logStarted(getApplicationLog(), stopWatch);
            }
            listeners.started(context);
            callRunners(context, applicationArguments);
        }
```

```
		catch (Throwable ex) {
			handleRunFailure(context, ex, exceptionReporters, listeners);
			throw new IllegalStateException(ex);
		}

		try {
			listeners.running(context);
		}
		catch (Throwable ex) {
			handleRunFailure(context, ex, exceptionReporters, null);
			throw new IllegalStateException(ex);
		}
		return context;
	}
```

1

1

1 1

1